

Alternatives to Re-Planning: Methods for Plan Re-Evaluation at Runtime

Emmanuel Benazera

RIACS, NASA ARC, Moffet Field, CA 94035
ebenazer@email.arc.nasa.gov

Abstract

Current planning algorithms have difficulty handling the complexity that is due to an increase in domain uncertainty, and especially in the case of multi-dimensional continuous spaces. Therefore, they produce plans that do not take into account numerous situations that can occur at runtime, such as faults or other changes in the planning domain itself. Thus there is a gap between the plan generation and the reality experienced at runtime. Here we present two methods that allow the plan conditionals to be revised w.r.t. uncertainty on the system as estimated at runtime.

Introduction

The need for autonomy and robustness in the face of uncertainty is growing as planetary rovers become more capable and as missions explore more distant planets. Recent progress in areas such as instrument placement (Pedersen *et al.* 2003; 2005) makes it possible to visit multiple rocks in a single communication cycle. This requires reasoning over much longer time frames, in more uncertain environments. Simple unconditional plans as used by the Mars Exploration Rovers (MER) will probably have a low probability of success in such context, so that the robot would spend almost all its time waiting for new orders from home.

In the last decade, architectures for future planetary rover missions include a planner/scheduler, a health monitoring system, and an executive. The planner/scheduler generates a control program/plan that describes the sequence of run-time actions necessary to achieve mission goals. Since the rover's environment is highly uncertain (Bresina *et al.* 2002), the control programs (also called *plans*) are contingency plans (Dearden *et al.* 2003) in that they involve conditional branches that are based on decision functions of the system state that the executive can evaluate in real time. The executive is responsible for the execution of the control programs, taking into account the current state of the system as estimated by the health monitoring system. This capability includes deciding the best branch in a plan when reaching a branch point, given an estimate of the current system state, inserting and replacing plan portions to react to faults and other unpredictable events.

However, planners have difficulties handling certain situations, such as actions that carry no utility (typically used for responding to unlikely situations) and fault occurrences, or to

prepare for a belief state update¹. First, actions with no reward can possibly be inserted anywhere in the plan at low cost, so the greedy approach that seeks to maximize the expected utility fails to position them efficiently. Second, planner domains describe a very limited set of faults, thus relying on a mostly nominal model of the world and system actions (e.g. no stuck wheels, broken navigation system, rocky environment,...). Moreover, fault models exponentially increase the complexity of the planning even if the faults have low probability of occurrence as they can occur at any time during the plan execution. Finally, the health monitoring system returns an ever changing belief state over time that has to be taken into account. For these reasons, the response to unlikely situations and faults is better decided at execution: the health monitoring system passes a belief over the system state to the executive that decides which portion of the plan to execute, sometimes inserting/replacing wanted/unwanted plan blocks.

More recent architectures try to mitigate these problems by moving towards unified planning and execution frameworks (Alami *et al.* 1998; Muscettola *et al.* 2002; Estlin *et al.* 2005). Several of these architectures are discussed at the end of this paper, however it is well understood that uncertainty in future values forces an agent to plan locally. For example, to mitigate this problem, (Muscettola *et al.* 2002) allows plans to include explicit calls to a deliberative planner. This comes back to finding place where to insert a branch, and as demonstrated in (Dearden *et al.* 2003), the branch point is usually not situated at the point that has the highest probability of failure. Now note that if the process of estimating a good branching point does not forcibly require to do the planning, it doesn't cost much to pre-plan the branch once the point has been identified. Therefore, the branch can be pre-planned and its values later updated during execution. As it will be explained later in this paper, re-evaluation of a plan is in no way equivalent to re-planning, but a re-evaluated plan can be found that is optimal w.r.t. the information on the uncertain system state and the original plan.

We said that most planners do not handle well the complexity due to the presence of faults in a model and therefore rarely include faults within their planning domain. Moreover, major faults are well known and recoveries can be efficiently con-

¹Partially Observable Markov Decision Processes (POMDPs) allow the latter but are often untractable.

structured before execution. At runtime, a fault detection system, or more generally, a state estimator will return a state estimate that triggers one or more plan fragments for system recovery or opportunistic science. These plan fragments are often referred to as floating contingencies whose execution can be conditioned upon resources (including time) and/or system behavioral modes. Therefore in this paper we will refer to two types of contingencies: pre-planned branches on resources that are part of the main plan, and floating contingencies, that trigger in response to certain events and resource values. The paper focuses on techniques to re-evaluate the former, and studies the complexity added to them by the latter.

The problem can be seen as one of re-evaluating the plan values, such as its utility, and updating the plan conditionals, i.e. the branch conditions. Typically, at runtime, the probability mass of the state estimate shifts among regions of the hybrid space (continuous resources plus discrete state). We adapt the pre-computed branch conditions to these changes by projecting the changes forward and backing up the resulting states. Our first approach is an adaptation of the classical Monte Carlo (MC) technique (Sutton & Barto 1998; Thrun 2000). Our second approach is based on decision theoretic techniques and converts the problem into a small Partially Observed Markov Decision Problem (POMDP) (see (Kaelbling, Littman, & Cassandra 1998) for an introduction and more references) whose solving at runtime returns probabilistic decision lines that are optimal given the initial plan.

Preliminaries

Here a plan can be seen as a tree whose nodes are known as the branch points. The value function for a node is a continuous function over the multi-dimensional resource state, i.e. a mapping from the resource space to the utility space, and that depends on downstream node value functions. Planning determines a set of policies that maximize the expected utility of the plan. At branch points, this leads to conditions over the resource space that discriminate among branches.

Typically, planning proceeds to a mapping from the system state space to the utility space, i.e. the utility obtained by executing the plan, that it seeks to maximize. Noting the system state $s = (x, r)$ with $x \in X$ the discrete state (or system modes), and $r \in R$ the multi-dimensional continuous state (including time), the utility earned by executing a branch b_i starting at s can be noted:

$$V_{b_i}(s) = \sum_{x' \in X} \int_R p((x', r') | s, a_{i1}) [U(a_{i1}, (x', r')) + V_{B_i}(x', r')] dr' \quad (1)$$

with a_{i1} the first action of branch b_i , B_i the remaining portion of the branch, $U(a_{i1}, (x', r'))$ the utility earned, and s' the system state after executing a_{i1} following the probability distribution $p(s' | s, a_{i1})$. Over a belief state $\pi(s)$, as estimated by the health monitoring system, we have:

$$V_{b_i}(\pi(s)) = \sum_{x \in X} \int_R V_{b_i}(x, r) \pi(x, r) dr \quad (2)$$

And at a branch point where n branches are available, the best branch is decided according to:

$$b^* = \arg \max_{i \in [1, n]} V_{b_i}(\pi(s)) \quad (3)$$

This is similar to the Bellman equations for POMDPs (Boyan & Littman 2000). Each value function $V(b)$ maps the resource space to the utility of the branch b . The max operator of relation (3) defines an upper bound on the branch point overall utility value, and branch conditions are found at the functions intersections. At execution, deviations from the planning domain and information of the state estimate move these decision lines.

There are several conditions and situations under which the plan value must be re-evaluated. First, when the execution encounters a branch point, any change in the Bellman equation functions, such as the belief b over the state s , the reward model U , the action cost model, requires that all branch functions at this branch point are re-evaluated. Second, if not at a branch point, but if a floating branch has to be inserted, then the plan equation is changed and the remaining portion of the currently executed branch as well all future branch conditions must be re-evaluated. For example, when inserting a branch b_f , equation (1) becomes:

$$V_{b_f}(s) = V_{b_f}(s) + \sum_{x' \in X} \int_R p((x', r') | s, b_f) V_B(x', r') dr' \quad (4)$$

where B is the remaining portion of the current plan to be executed after b_f . The local value of b_f is the expected reward from the actions within the floating branch itself. The remaining term is a representation of the end state of the local plan, including the probability of the resources remaining after executing the local plan.

The remaining of the paper studies approaches to the fast re-evaluation of these decision lines.

The Monte-Carlo Approach to the Re-Evaluation of Contingency Plans

Approximating branch average utility

Applying Monte Carlo techniques to the approximation of equation (2) is straightforward: the integral over the multi-dimensional continuous space is turned into a sum by sampling N times from $b(s)$ and $p(s' | s, a)$, and the utility is averaged over the successive runs. We note:

$$\hat{V}_{b_i}(\pi(s)) = \sum_{x \in X} \sum_{x' \in X} [U(a_{i1}, s'_j) + \hat{V}_{B_i}(s'_j)] \quad (5)$$

where $s'_j \sim p(s' | s_j, a_{i1})$ and $s_j \sim b(s)$. The larger the N , the better the fit to the underlying probability distributions, and the better the approximation.

Plan simulation

For simulating branches with MC, we use a prioritized pile of events including plan actions, and a set of constraints among them. The pile is filled up with actions whose execution is simulated by testing their temporal constraints and sampling their consumption before being rewarded and popped out.

Sampling decisions

We sample the decision by deciding the path with highest utility for each sample. We write:

$$\hat{V}^{dec}(\pi(s)) = \frac{1}{N} \sum_{j=1}^N \max_{i \in [1, n]} \hat{V}_{b_i}(\pi(s)) \quad (6)$$

In algorithm 1, each path is explored by each sample for the

- 1: **for all** $j < N$ **do**
- 2: Proceed with MC on the first branch.
- 3: **for all** branches b_i at branch point **do**
- 4: Apply this algorithm recursively to b_i , with $j = 1$.
- 5: Return the highest utility at this branch point (max).
- 6: Return the averaged utility of the plan.

Algorithm 1: Recursive procedure for sampling decisions

evaluation of the max operator. The averaged returned utility is near optimal, but the sampled decision for the best branch (the arg operator) depends on the sampled resource space that must be partitioned into subregions of identical decision.

Floating contingencies

Floating contingencies are a challenge to the simulator because they can trigger at anytime. The simulator uses random events to trigger these branches and specific dynamic constraints to handle their insertion. The complexity increase due to floating branches is a product of the number of plan actions, actions in the branch, and the number of these branches. The next section covers the retrieval of the decision lines in the multi-dimensional resource space.

Bounding the resource space for deciding future branches

Decision at branch points can be made based on the simulation results by executing the branch with the highest earned utility average. Simulation provides sufficient information for computing branch conditions at future branch points. This operation is performed at virtually no cost and can spare future simulations by constraining future decisions.

Approximating branch decision lines thru piecewise constant value function approximation Our solution is to slice the resource domain into rectangular bins and to fit the branch value functions in each bin with a piecewise constant function, based on the MC samples. Function intersections are found at bin edges. Noting Δ_r a bin in the resource space, we can write b_i 's value:

$$\hat{V}_{b_i}(\pi(s)) = \sum_{\Delta_r} \sum_{x \in X} p(b_i | \Delta_r) p(\Delta_r) \hat{V}_{b_i}(\Delta_r, x) \quad (7)$$

i.e. as the sum of the average utilities of b_i in each bin when it is the branch with the highest expected utility. More precisely:

$$\hat{V}_{b_i}(\Delta_r, x) = \frac{1}{n_{r\Delta_r}} \sum_{r_j \in \Delta_r} \sum_{x' \in X} \hat{V}_{b_i}(s_j) \quad (8)$$

with $s = (x, r)$ and $s_j = (x', r_j)$, is the average utility of b_i on bin Δ_r from the $n_{r\Delta_r}$ samples r^j it contains,

$$p(b_i | \Delta_r) = \frac{1}{n_{r\Delta_r}} \sum_{r_j \in \Delta_r} \delta(b_i = \arg \max_{i \in [1, n]} \hat{V}_{b_i}(\pi(s_j))) \quad (9)$$

where δ is the Dirac function, is the probability for b_i to be the branch with the highest utility over the samples of the bin,

$$p(\Delta_r) = \frac{n_{r\Delta_r}}{N} \quad (10)$$

is the probability of the bin itself. An optimal bin size W is

- 1: Proceed with algorithm 1 and collect samples at branch point.
- 2: **for all** branch points in the contingency plan **do**
- 3: Compute the optimal bin size and slice the space into bins.
- 4: Compute statistics with equations (8), (9) and (10).
- 5: Evaluate equation (7) for each branch.
- 6: In each bin, identify the branch with the highest value.
- 7: Identify new branch conditions where successive bins have different highest utility branches.

Algorithm 2: Branch conditions approximation thru piecewise constant value function approximation

obtained, in the sense that it provides the most efficient unbiased estimation of the probability distribution function formed by the samples. We used $W = 3.49\sigma N^{-1/3}$ where σ is the standard deviation of the distribution, here estimated from the samples (D. 1976; A.J. 1991). The overall strategy is presented on algorithm 2.

Branch conditions are obtained by comparing the branch with the highest utility for each bin: if two successive bins return different results, a branch condition exists at their edge. Thus, the precision of the approximation is directly dependent on the optimal bin size, that depends on the number of samples. Stutter at decision point can be overcome by fitting the successive piecewise constant approximations with more smoothly curve.

Belief update on re-evaluated branch conditions The re-evaluated decision functions are inequalities of the form $r' \leq (\geq)g(r)$. Given a state estimate $\pi(s)$ at branch point, decision over n branches follows:

$$\begin{aligned} b^* &= \arg \max_{i \in [1, n]} \sum_{x \in X} \int_{r \leq g(r)} V_{b_i}(x, r) \pi(x, r) dr \\ &\approx \arg \max_{i \in [1, n]} \sum_{x \in X} \sum_{\Delta_{r'}} p(b_i | \Delta_{r'}) p(\Delta_{r'}) \hat{V}_{b_i}(\Delta_{r'}, x) \pi(r \in \Delta_{r'}) \end{aligned}$$

with r' such that $\forall r' \in \Delta_{r'}, r' \leq g(r)$.

Discussion

The major drawback of the Monte-Carlo approach is that it provides a probabilistic guarantee of its results, that is never absolute. This is a problem that we partially address in the next

section with the use of a decision theoretic formulation. Another work, (Jain & Varaiya 2004) finds bounds on the number of samples for the convergence of the expected reward for a class of policies.

Decision theoretic approach to plan re-evaluation

Another problem with the MC approach is that the decision is made based on a mapping from the continuous resource space to the utility space that forces the approximation of the decision lines. An alternative is to use a mapping from the belief space over the decisions to the utility space. The decision space is finite, made of the branch conditions of the original plan. The belief space over the decision is continuous and of dimension the number of decisions minus one. This formulation leads to an enlarged space but allows the use of decision theoretic techniques to directly incorporate the belief space in the computation of optimal decision lines. More precisely our problem can now be casted into a small POMDP whose actions are the plan branches, the states the branch conditions, the observations the system states.

Plan reduction to a POMDP

A standard POMDP is made of a set of actions, a set of states, a set of transitions among states per action, and a set of observations. In our model, we abstract away the actions and use a branch an action for the POMDP. Our POMDP is then defined as a tuple (F, S, B, L, T, R) where:

- F is a finite set of branch decision outcomes (as states),
- S is a finite set of system states (as observations),
- B is finite set of branches (as actions),
- $P(s | b, f')$ is the probability of state s given that branch b has been executed and has landed in f' ,
- $P(f' | b, f)$ is the probability of entering outcome f' after taking branch b in outcome f ,
- $R(f, b)$ is the reward for taking branch b while in outcome f .

The POMDP belief update can be expressed as:

$$\pi_b(f', s) = \frac{P(s | b, f') \sum_{f \in F} P(f' | b, f) \pi(f)}{p(s | b, \pi)} \quad (12)$$

where π is a probability distribution (belief) over F , given s and b , and:

$$P(s | b, f') = \frac{P(f' | b, s) p(b, s)}{p(f')} \quad (13)$$

The value of executing branch b under decision f and state s is:

$$V(f, s) = R(f, b, s) + \gamma \sum_{f' \in F} P(f' | b, f) \sum_{s' \in S} P(s' | f', b) V(s', f') \quad (14)$$

where in the absence of floating contingencies (because f can only lead to b):

$$P(f' | b, f) = P(f' | b) = \sum_{s' \in S} P(f' | b, s') p(s') \quad (15)$$

and $R(f, b, s) = V_b(b(s))$, from equation (2). Finally the value of executing branch b from some belief state π and observing s is:

$$V_s(\pi_b) = \sum_{f \in F} \pi(f, s) V(f, s) \quad (16)$$

and the optimal value function is given by:

$$V(\pi) = \max_{b \in B} \sum_{s \in S} p(s) V_s(\pi_b) \quad (17)$$

Simulation

The successor states s' and the $p(s')$ of equation (15) are unknown and must be obtained through simulation. As a simulator we use the MC algorithm of the previous section and generate both the $\hat{V}_b(s)$ and the s' in a depth first forward search in the plan tree.

Solving

The solving of this POMDP returns a piecewise linear convex value function that is a mapping from the belief space over the decision outcomes to the highest expected plan utility. Optimal branch conditions are found at the intersections of maximized value functions and are now conjunctions of inequalities of the form $P(r \leq h(r)) \leq c$ where $r \leq h(r)$ is the branch condition from the original plan and c a constant in $[0, 1]$. For any belief over an outcome, the solution returns the optimal policy, w.r.t. the original plan.

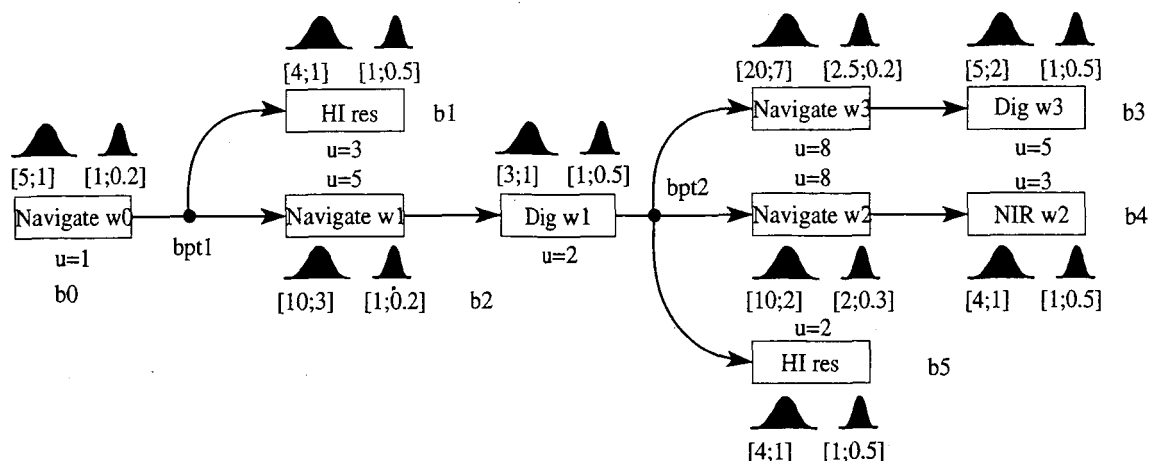
Floating contingencies

Floating contingencies pose a serious problem to the decision theoretic approach because the possible interruption of any action within a branch leads to a potentially infinite number of actions (breaking up a branch an infinite number of times over resource and time values with non null probability). Approaches like (Younes & Simmons 2004) can be used here to handle the asynchronous events, but do not allow to include the events (here floating contingencies) within the policy (therefore the computation of their conditions is not possible). While we are not yet sure about the range of solutions to this problem, it seems realistic to research approximations of floating conditions over a single branch.

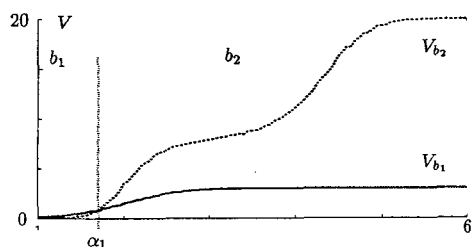
Results

A contingency plan for the Mars exploration domain

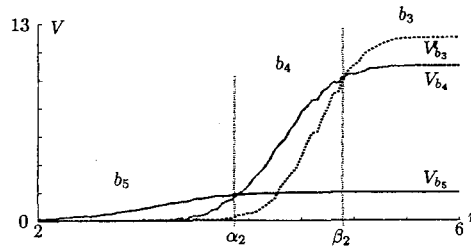
Our application is on a planetary rover plan. Consider the plan for a Mars rover on figure 1. It tells the rover to first navigate to a waypoint w_0 , and there to decide whether to take a high resolution image of the point (HI res) or to move forward to a second waypoint w_1 depending on the level of resources (here energy and time). After reaching w_1 and digging in the soil, it



(a) Contingency plan for the Mars rover domain



(b) Value functions of branches at branch point 1 (bpt1)



(c) Value functions of branches at branch point 2 (bpt2)

Figure 1: Branch value functions at branch point for a detailed rover problem

must decide whether to move forward to waypoints w_3 or w_2 or to simply get an image at w_1 and wait for further instructions. NIR is a spectral image of a site or rock. Action time and energy consumptions are represented as Gaussian bumps of empirical mean and variance. In this example branch conditions at branch points $bpt1$ and $bpt2$ have the following parameters: $\alpha_1 = 0.1$, $\alpha_2 = 2.1$ and $\beta_2 = 2.2$.

Decision sampling

Branch conditions re-evaluation at branch points: bounds/bins are generated with the sampling decision algorithm, and verified by running a classical Monte-Carlo simulation, that does not maximize the utility, but follows the new branch conditions and averages the earned utility. Simulation also returns the failure probability of the plan. The error is the difference to the optimal plan value in percentage. The piecewise constant approximation of the branch value functions returns good utility (Table 1). Figure 2 pictures results for the second branch point of our rover problem (the energy is pictured and the time line is omitted) and shows the shifting branch conditions on the horizontal axis that is the energy line.

N	Value	Time	V dec	err
100	14.21	0.03	10.9	23.3
500	13.618	0.16	9.732	28.53
2500	13.8244	0.78	11.2992	18.26
12500	13.8008	4.08	11.9542	13.27
62500	13.7835	20.79	12.156	11.8
312500	13.7717	120.3	12.1214	12
500000	13.777	223.89	12.1814	11.58

Table 1: Monte-Carlo decision sampling and branch condition re-evaluation based on MC samples. Results are as follows: N is the number of samples, V is the mean expected highest value obtained for the plan, V_{dec} is the value obtained when using the re-evaluated branch conditions, err the error percentage to the simulated best value.

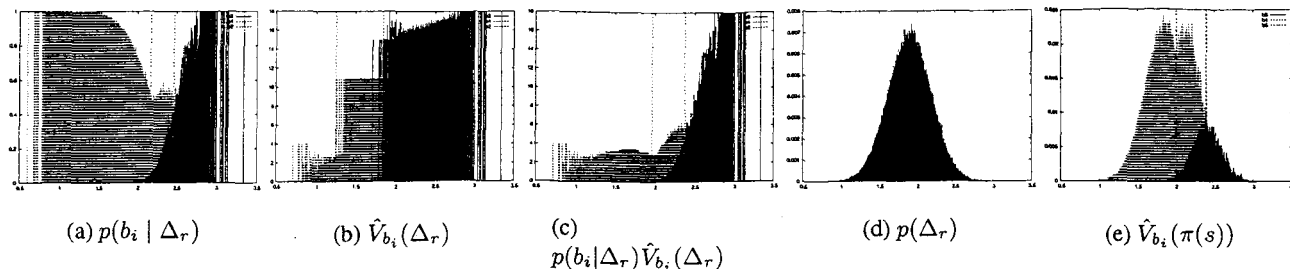


Figure 2: Piecewise constant approximation of branch value functions from simulation samples.

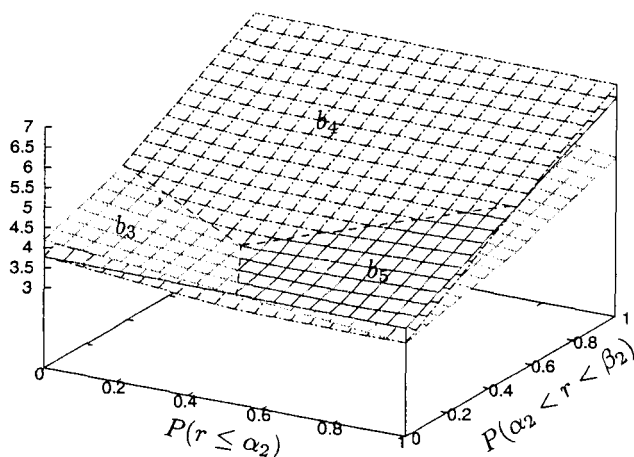


Figure 3: Optimal value function over the original plan branch conditions: x and y axis represent the probability of decision outcomes $P(r \leq \alpha_2)$ and $P(\alpha_2 \geq r \leq \beta_2)$. $P(r \geq \beta_2)$ is deduced from them.

Decision theoretic approach

We converted our example to a POMDP and simulated the observations and rewards, respectively the system states and branch value functions. Starting from a fixed level of resources, figure 3 shows the convex value function solution for the second branch point (bpt_2).

Comparison and Discussion

To compare the two approaches, we moved a gaussian belief of fixed variance 0.1 along the resource (energy) line and studied the decision for each resource value. Results are presented on figure 4. V_{mc} and dec_{mc} respectively denote the value obtained and the decision based on the Monte-Carlo method with $b_5 = 1$, $b_4 = 2$ and $b_3 = 3$; V_{dtp} and dec_{dtp} are based on the decision theoretic planning (dtp) approach. First, the difference in value between the two methods is due to the high level of branch failure (i.e. resource gets to zero) in the simulation for the decision theoretic approach (since it is based on the original branch conditions). This is of medium importance only when we study the decision making: we observe

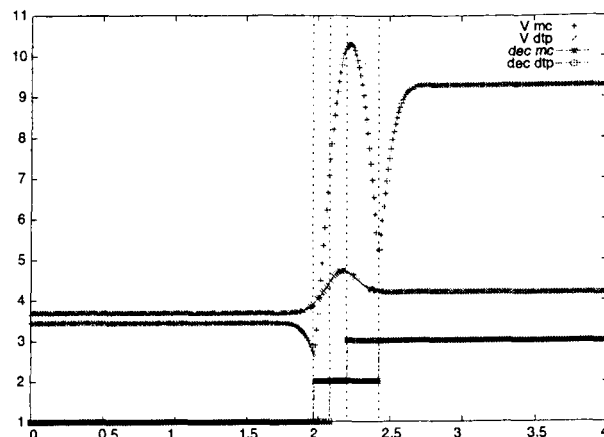


Figure 4: Comparison between the Monte-Carlo and the decision theoretic re-evaluation methods.

that decision to go to branch b_4 then to b_3 based on MC are respectively slightly early and slightly late, and this is visualized as two sudden drops in expected value; the decision based on dtp switches later to b_4 , and earlier to b_5 right at the highest value point. Overall the dtp-based decision leaves less room for branch b_4 , which can be surprising when looking at the large surface corresponding to b_4 on figure 3 but is explained by the fact that a high probability for on decision $2.1 < r < 2.2$ denotes a more accurate belief (given the fixed variance) than for other branches. Finally it is difficult to fully assess the dominance of one method over the other. At this point of our research we lean in favor of the MC approach for plans with a small number of actions and a high number of decision outcomes, and for the dtp approach when a high number of actions in a plan makes the price of successive MC simulations costly.

Related Work and Conclusion

We have presented a simple strategy for the robust execution of contingency plans under uncertainty. It re-evaluates branch value functions at branch point and re-estimates branch conditions whenever necessary. The framework allows runtime insertion/replacement of plan portion thru the use of floating

branches but much work on their full integration into the re-evaluation process should follow. This is the first step towards the development of more powerful techniques for planning and execution under uncertainty. The MC approach is flexible and provides good results in any situation given that a sufficiently high number of samples is used. The algorithms presented are a baseline capability, and will be used later to assess the quality of more complex and focused approaches.

Related Work

Other works on plan re-evaluation include (Gough, Fox, & Long 2004) that studies plan execution with uncertainty on the resource consumption. However, the executed plans are no contingency plans as branch execution is not conditioned on decision functions over the resource state. (Washington & Lees 2004) develops a fast method for plan portions insertion/replacement, but partly fails here as it relies on pre-computed value functions (this is not always possible as faults change the model of actions).

Mixed planning/execution include (Alami *et al.* 1998) that uses a deliberative planner and an executive on top of a set of reactive controllers. (Estlin *et al.* 2005) presents the Closed-Loop Execution and Recovery (CLEaR) system that is intended to run on rovers with little communication with ground. CLEaR closely integrates the CASPER continuous planner (Chien *et al.* 2000) and the TDL executive system (Simmons & Apfelbaum 1998). Plan re-evaluation and the methods described in this paper can be seen as an alternative to the iterative plan repair of CASPER. We view plan re-evaluation as an intermediate step between execution of pre-planned contingencies and re-planning. Re-planning will always be necessary as if a situation occurs on-board for which there is no pre-planned contingency, the rover must wait for instructions. In that sense, plan re-evaluation complements architectures such as (Muscettola *et al.* 2002) and (Estlin *et al.* 2005).

For solving the decision theoretic problem, fast techniques such as (Feng & Zilberstein 2004) allow the solving of rather large problems. Given we abstract away actions within the branches when formulating the POMDP, we see our problems (not including the floating contingencies) as being of a small size.

Future Work

Future work includes dealing with floating contingencies within the decision theoretic framework, pre-computing more advanced branch value functions at planning time (Feng, Meuleau, & Washington 2004) and using them at runtime. Another hot topic remains the re-evaluation of plans that contain concurrent actions. Also note the new class of problems recently arisen in the rover domain, where the robot is able to satisfy only a subset of the goals (Smith 2004). In that case, re-evaluating the plan is not as efficient anymore because the change in resource consumption would in general lead to the selection of a different subset of goals.

Acknowledgements

Ideas in this paper are based on the experience and work of a group of current and past researchers at NASA Ames Research

Center. The author thought it was time to bring some of these ideas to life and share the accumulated experience, and thanks R. Washington, R. Dearden, S. Narasimhan, H. Cannon, T. Willeke and D. Roland.

References

- A.J., I. 1991. Recent developments in non parametric density estimation. *Journal of the American Statistical Association* 413(86):205–224.
- Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research* 17(4).
- Boyan, J., and Littman, M. 2000. Exact solutions to time-dependent mdps. In *Advances in Neural Information Processing Systems 13*, 1–7.
- Bresina, J.; Dearden, R.; Ramkrishnan, S.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for ai. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*.
- Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 2000. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Breckenridge, CO.
- D., S. 1976. On optimal and data-based histograms. *Biometrika* 66:605–610.
- Dearden, R.; Meuleau, N.; Ramkrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *ICAPS-03: Proceedings of the Workshop on Planning under Uncertainty and Incomplete Information*, 415–428.
- Estlin, T.; Gaines, D.; Chounard, C.; Fisher, F.; Castano, R.; Judd, M.; Anderson, R.; and Nesnas, I. 2005. Enabling autonomous rover science through dynamic planning and scheduling. In *to appear in IEEE Aerospace 2005*.
- Feng, Z., and Zilberstein, S. 2004. Region-based incremental pruning for pomdps. In *20th Conference on Uncertainty in Artificial Intelligence (UAI-04)*, 146–153.
- Feng, Z.; Meuleau, N.; and Washington, R. 2004. Dynamic programming for structured continuous markov decision problems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*.
- Gough, J.; Fox, M.; and Long, D. 2004. Plan execution under resource consumption uncertainty. In *Proceedings of the Workshop on Connecting Planning Theory with Practice at ICAPS-04*, 24–29.
- Jain, R., and Varaiya, P. 2004. Simulation-based value function estimates of discounted and average-reward mdps. In *Proceedings of the Conference on Decision and Control, 2004*.
- Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.
- Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; and Plaunt, C. 2002. Idea: Planning at the core of autonomous reac-

tive agents. In *in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Pedersen, L.; Bualat, M.; Lees, D.; Smith, D.; and Washington, R. 2003. Integrated demonstration of instrument placement, robust execution and contingent planning. In *Proceedings of the 7th Int. Symp. on Artificial Intelligence, Robotics and Automation in Space*.

Pedersen, L.; Smith, D.; Deans, M.; Sargent, R.; Kunz, C.; Lees, D.; and S.Rajagopalan. 2005. Mission planning and target tracking for autonomous instrument placement. In *Submitted to 2005 IEEE Aerospace Conference*.

Simmons, R., and Apfelbaum, D. 1998. A task description language for robot control. In *Proceedings of the Intelligent Robots and Systems Conference, Vancouver, CA*.

Smith, D. 2004. Choosing objectives in over-subscription planning. In *Proceedings of ICAPS-04*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998.

Thrun, S. 2000. Monte carlo POMDPs. In Solla, S.; Leen, T.; and Müller, K.-R., eds., *Advances in Neural Information Processing Systems 12*, 1064–1070. MIT Press.

Washington, R., and Lees, D. 2004. Utility-based plan insertion for continuous resources. In *Proceedings of the IEEE 2004 International Conference on Robotics and Automation*.

Younes, H., and Simmons, R. 2004. Solving generalized semi-markov decision processes using continuous phase-type distribution. In *In Proceedings of the Nineteenth National Conference on Artificial Intelligence - AAAI-04*.